# Algoritmid ja andmestruktuurid

ja nende keerukus

Marko Kääramees

# Miks on vaja algoritme ja andmestruktuure

Leida kiireim tee punktist A punkti B

- Andmestruktuurid

  Andmed tuleb kuidagi esitada

  – Sõidurajad, võimalikud pöörded, mitmetasandilised teed, ühesuunalised tänavad, liikumiskiiruse erinevus


- Kiire algoritm
  – Piiratud arvutusvõimsus
  – Suur andmemaht (kõik Euroopa teed-tänavad)
  – Arvuti ei vaata kaardile “ülevalt-alla” arvutada tuleb andmebaasi kirjetel, mitte graafilisel pildil

# Kuidas sorteerida mullitajaid?

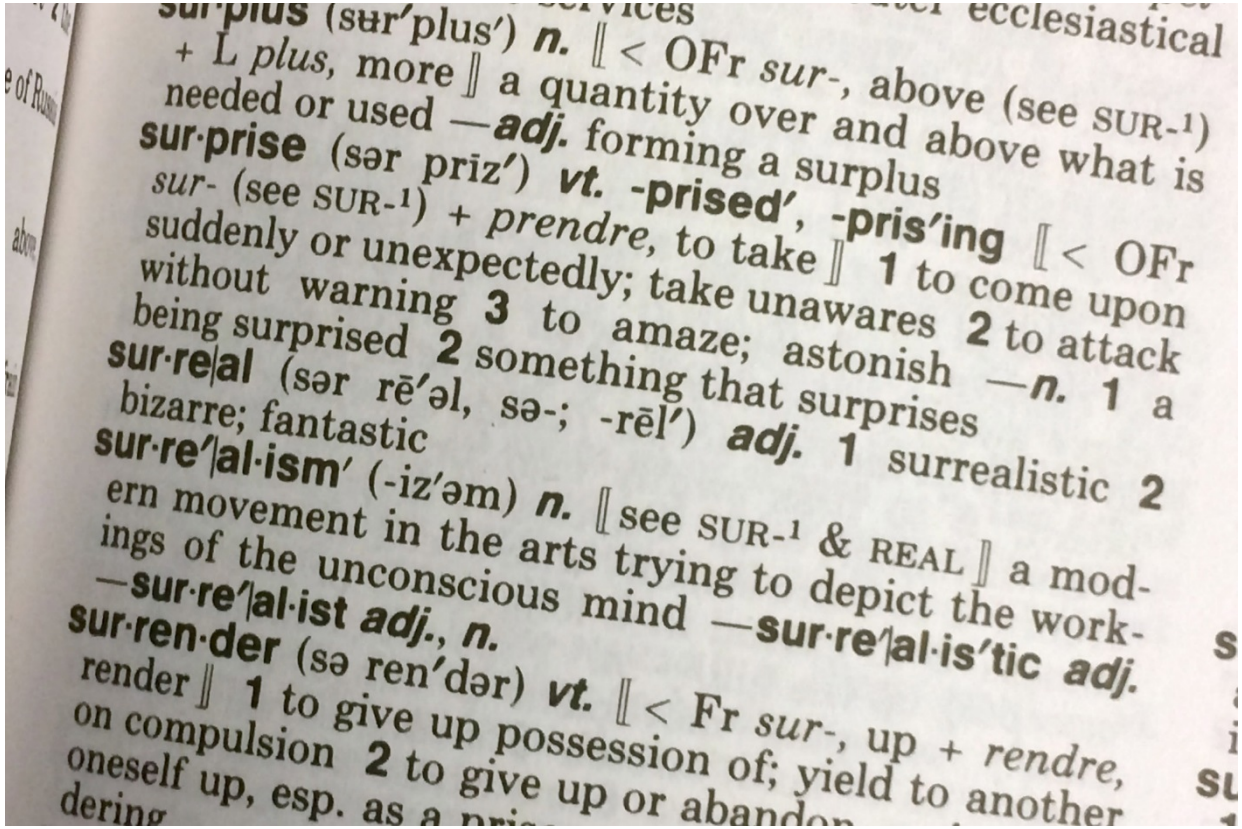
Reeglid:

See mullitaja mida kätte võtad, tuleb panna kaalule

Võidab see, kes kaalub kõige vähem kordi

# Miks sorteerida?

- Otsimine põhineb sellel, et andmed on sorteeritud



**sur·plus** (sʉr′plus′) ***n.*** ‖ < OFr *sur-*, above (see SUR-1) + L *plus*, more ‖ a quantity over and above what is needed or used —***adj.*** forming a surplus

**sur·prise** (sər priz′) ***vt.*** **-prised′**, **-pris′ing** ‖ < OFr *sur-* (see SUR-1) + *prendre*, to take ‖ **1** to come upon suddenly or unexpectedly; take unawares **2** to attack without warning **3** to amaze; astonish —***n.*** **1** a being surprised **2** something that surprises

**sur·re|al** (sər rē′əl, sə-; -rēl′) ***adj.*** **1** surrealistic **2** bizarre; fantastic

**sur·re′|al·ism′** (-iz′əm) ***n.*** ‖ see SUR-1 & REAL ‖ a modern movement in the arts trying to depict the workings of the unconscious mind —**sur·re′|al·is′tic** ***adj.*** —**sur·re′|al·ist** ***adj.***, ***n.***

**sur·ren·der** (sə ren′dər) ***vt.*** ‖ < Fr *sur-*, up + *rendre*, render ‖ **1** to give up possession of; yield to another on compulsion **2** to give up or abandon oneself up, esp. as a

# Selection sort

Korda kuni kõik on sorteeritud

leia minimaalne listi sorteerimata osast

tõsta see listis järgmise sorteerimata kohale

```
SelectionSort(list):
list : array of items
n : size of list
for i = 1 to n - 1
    min = i
    for j = i+1 to n
        if list[j] < list[min] then
            min = j;
if indexMin != i then
    swap list[min] and list[i]
```

# Mitu võrdlust on vaja?

- List 4 elemendiga
  - Min 4 elemendist – 3 võrdlust
  - Min 3 elemendist – 2 võrdlust
  - Min 2 elemendist – 1 võrdlust
  - Kokku 3+2+1 = 6 võrdlust
- List 8 elemendiga
  - Kokku 7+6+5+4+3+2+1 = 28 võrdlust
- Üldjuhul *n* elemendi korral aritmeetiline jada *n*-1

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2}$$

| | | | |
|---|---|---|---|
| **3** | **1** | **4** | **2** |
| **1** | **3** | **4** | **2** |
| **1** | **2** | **4** | **3** |
| **1** | **2** | **3** | **4** |

| *n* | võrdlusi |
|---|---|
| 4 | 6 |
| 8 | 28 |
| 16 | 120 |
| 32 | 496 |

# Algoritmide keerukus

**Algoritmi keerukus on põhioperatsiooni(de) arvu sõltuvusfunktsioon K(*n*) sisendi(te) suurusest *n*.**

- *Põhioperatsioon* ei ole üheselt defineeritav

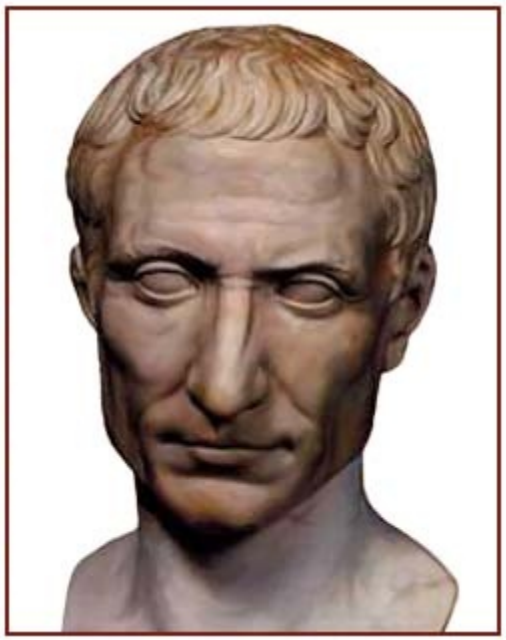
  Midagi mis on riistvaras tehtav piiratud arvu sammudega

  – aritmeetika tehe, võrdlus, omistus
  – vahel valitakse üks põhioperatsioon ja loetaks selle arvu, näiteks tsüklitingimuse võrdlus
  – mõnikord kasutatakse ka ridade arvu
- *Sisendi suurus* võib olla defineeritud erinevalt
  – Sisendandmete maht (massiivi, faili, andmebaasi suurus)
  – Sisendparameetri väärtus
  – Sisendparameetri suurus (bittide/baitide arv)

Kas õnnestuks sorteerida vähema arvu võrdlemistega?
Efektiivsemalt?

# Divide et Impera

CAESAR



Napoleon

# Jaga ja Valitse strateegia

- Jaga

  Jaga ülesanne lihtsamateks alamülesanneteks

- Valitse

  Lahenda alamülesanded

- Ühenda

  Leia lahendus alamülesannete lahenduste kaudu

# Merge sort

1. Jagame massiivi pooleks

   Sorteerime mõlemad väiksemad massiivid
2. Ühendame sorteeritud massiivid üheks sorteeritud massiiviks võrreldes omavahel mõlema massiivi väiksemaid elemente
3. Jagamise lõpetamistingimuseks on 1 elemendiga massiiv
4. Ühe elemendiga massiivi võib lugeda sorteerituks, tagastame sellesama massiivi


# Mergesort

```
mergesort (int n, list S[ ]) {
   if (n>1) {
      int h=[n/2],       m = n - h;
      list U[1 ..h], V[1 ..m];
      copy S[1] through S[h] to U[1] through U[h];
      copy S[h+1] through S[n] to V[1] through V[m];
      mergesort(h, U);
      mergesort(m, V);
      merge (h, m, U, V, S);
} }
```

```
merge (int h, int m, list U[ ], list V[ ], list S[ ]) {
   index i, j, k;
   i = 1; j = 1; k = 1;
   while (i <= h && j <= m) {
      if (U[i] < V[j]) { S[k] = U[i]; i++; }
      else { S[k] = V[j]; j++; }
      k++; }
   if (i>h) copy V[j] through V[m] to S[k] through S[h+m];
   else copy U[i] through U[h] to S[k] through S[h+m]; }
```

# Mitu võrdlust on vaja?

- Tähistame keerukuse funktsiooniga K(n), kus n on sisendlisti elementide arv
- Tuleb sorteerida kaks osalisti pikkusega n/2, selle jaoks on vaja teha 2 korda K(n/2) võrdlust
- Tuleb kaks sorteeritud osalisti kokku panna (*merge*)
  - Tulemus on pikkusega *n*
  - Iga võrdlusega saame tulemusse 1 elemendi juurde, viimase niisama
    kokku *n*-1 võrdlust

$$K(n) = 2K(n/2) + (n-1)$$

???

# Proovime

- Kui listis on 2 elementi
  - 1 võrdlus
- Kui listis on 4 elementi, siis
  - Osalistide sorteerimised 1+1
  - Merge 3
  - Kokku 5
- Kui listis on 8 elementi, siis
  - Osalistide sorteerimised 5+5
  - Merge 7
  - Kokku 17
- Üldjuhul

$$(n\text{-}1)\log(n\text{-}1) + \log(n)$$

# Mis vahet neil on?

Eestis oli elanikke 1. jaanuar 2017

**1315635**

- Sorteerime need ära
  - Arvuti protsessor töötab 4 GHz
  - Iga taktiga tehakse üks võrdlus

| Algoritm | võrdlusi | Aega |
|---|---|---|
| Selection sort | $8.7*10^{11}$ | 216s = 3.5 min |
| Merge sort | $8.7*10^{7}$ | 0.007s |

# O-notatsioon

- O-notatsioonis esitatakse keerukusfunktsiooni määrav komponent

$$1/2\ (n\text{-}1)*n \in O(n^2)$$

- Liidetavatest suurima keerukusega funktsioon

$$n^2 - n \in O(n^2)$$

- Konstantseid kordajaid võib ignoreerida

$$\frac{1}{2}*n^2 \in O(n^2)$$

- Spetsiaalsed keerukusklassid:

| | |
|---|---|
| ***logaritmiline***: | **O(log *n*)** |
| ***lineaarne***: | **O(*n*)** |
| ***polünomiaalne***: | **O($n^k$), $k \geq 1$** |
| ***eksponentsiaalne***: | **O($a^n$), $n > 1$** |

# Sorteerimiste keerukused

- Selection sort

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2} \in O(n^2)$$

- Merge sort

$(n-1)\log(n-1) + \log(n) \in O(n \log n)$

Neid on võimalik määrata ilma katsetamata, algoritmi analüüsides

# Quicksort

1. Valime ühe elemendi teljeks (pivot)

   Sorteerime kõik teljest väiksemad temast vasakule ja suuremad paremale

   Sorteerime teljest paremale ja vasakule jääva massiivi
2. Ühendame vasaku massiivi, telje ja parema massiivi üksteise järgi
3. Jagamise lõpetamistingimuseks on 1 elemendiga massiiv
   Ühe elemendiga massiivi võib lugeda sorteerituks, tagastame sellesama massiivi

# Erinevad keerukusmõõdud

| n | Võrdlusi min | Võrdlusi max |
|---|---|---|
| 2 | 1 | 1 |
| 4 | 4 | 6 |
| 8 | 13 | 28 |

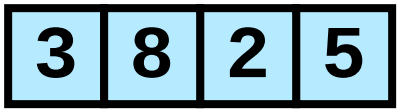
- Halvima juhu keerukus $O(n^2)$
- Keskmise juhu keerukus $O(n \log n)$

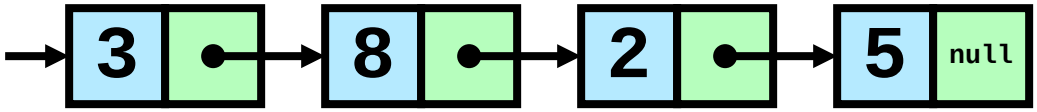
Mis on *quicksort-i* jaoks halvim sisend?

# Edasi?

- Kas saab veel kiiremini?
- Kui kasutada võrdlemist, siis ei saa

# Massiiv ja list



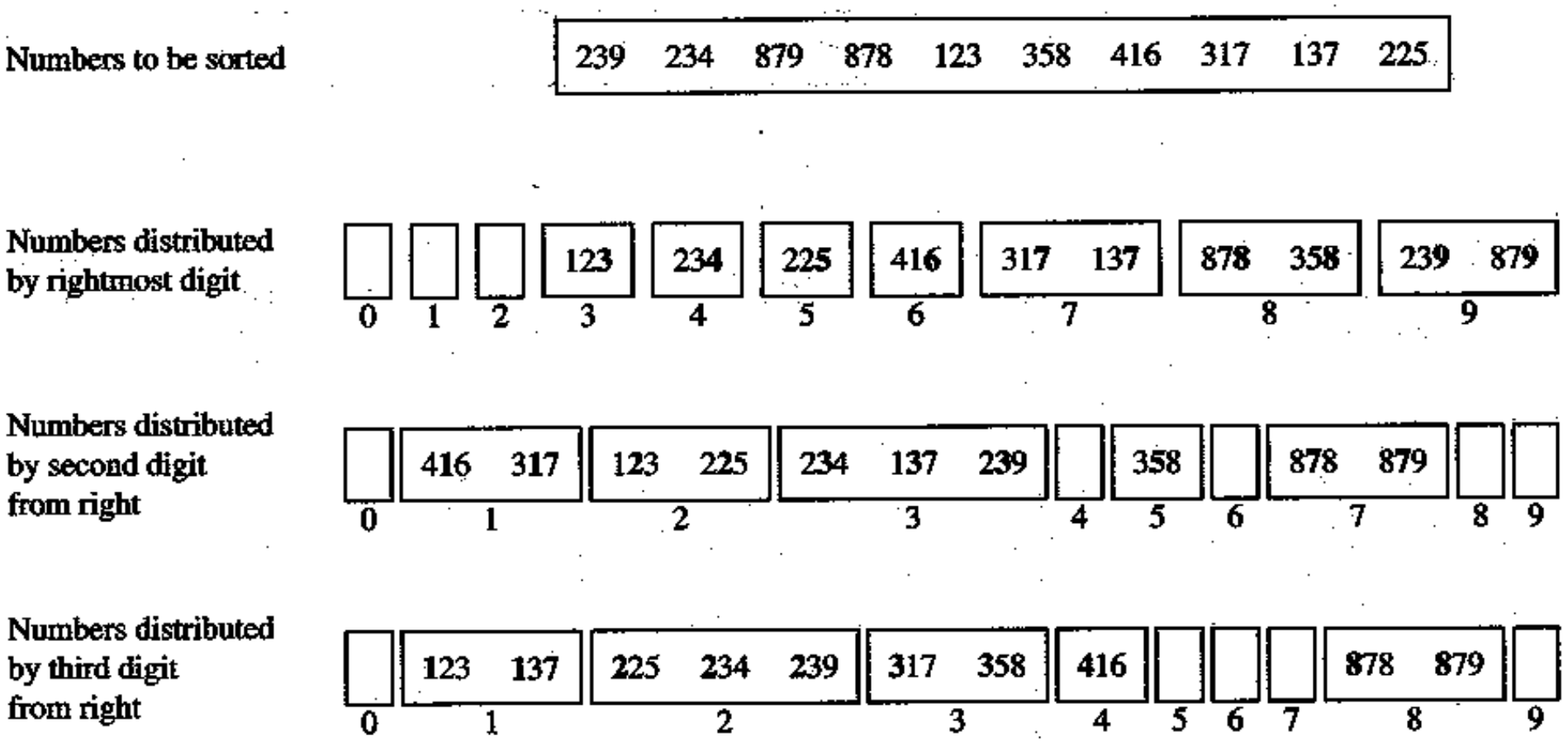

Massiiv

+ kiire pöördumine
- piiratud maht
- aeglane *add/delete*

List

- aeglane pöördumine
+ piiramatu maht
+ kiire *add/delete*

# Lingitud listi rakendus - Radix sort



# Radixsort

```
void radixsort (node_pointer& masterlist,
                int numdigits)
{
   index i;
   node_pointer list[0..9];

   for (i = 1; i <= numdigits; i++) {
      distribute(i);
      coalesce;
   }
}
```

# Radixsort

```
void distribute (index i)                          // i is index of current
{                                                  // digit being inspected.
   index j;
   node_pointer p;

   for (j = 0; j <= 9; j++)                        // Empty current piles.
      list[j] = NULL;
   p = masterlist;                                 // Traverse masterlist.
   while (p != NULL) {
      j = value of ith digit (from the right) in p-> key;
      link p to the end of list[j];
      p = p-> link;
   }
}
```

# Radixsort

```
void coalesce ()
{
   index j;

   masterlist = NULL;                                   // Empty masterlist.
   for (j = 0; j <= 9; j++)
      link the nodes in list[j] to the end of masterlist;
}
```

# Radixsort keerukus

- Põhioperatsiooniks *pointeri* omistamine
  - *distribute* *n*
  - *coalesce* 10
- Radixsort põhitsükkel
  - sõltub kohtade arvust *numdigits*

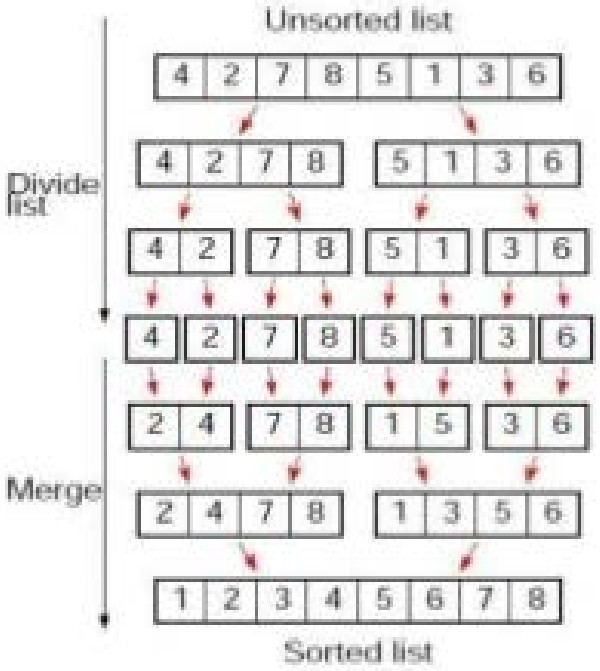
$$T(n) = numdigits\ (n + 10) \in \Theta(numdigits * n)$$

n = numdigits $T(n) \in \Theta(n^2)$

kõik sorteeritavad on erinevad $T(n) \in \Omega(n \log n)$

# Paralleelne Mergesort

```
ALGORITHM: mergesort(A)
1  if   (|A| = 1) then return A
2  else
3       in parallel do
4            L := mergesort(A[0..|A|/2))
5            R := mergesort(A[|A|/2..|A|))
6       return merge(L, R)
```




Keerukus O($n$), vajab O($n$) protsessorit

Odd-Even Parallel Mergesort võib sorteerida keerukusega O((log $n$)$^2$), vajab O($n$) protsessorit

# Kokkuvõtteks

- **Tasub mõelda**
  probleemi lahendades tasub leida efektiivne algoritm
- **Tasub analüüsida**
  Keerukust saab analüüsida algoritmi pealt ilma aega mõõtmata
- **Vee all võivad olla karid**
  Alati ei lange keskmine ja halvima juhu keerukus ei pruugi kokku langeda
- **Jõuga ja nõuga võib saada rohkem**
  Ülesande ümberdefineerimine ja paralleelsus aitavad